Owner Prediction for Accelerating Cache-to-Cache Transfer Misses in a cc-NUMA Architecture

Manuel E. Acacio, José González[†], José M. García and José Duato[‡]

Universidad de Murcia, Spain. E-mail: {meacacio,jmgarcia}@ditec.um.es
[†]Intel Barcelona Research Center, Intel Labs, Barcelona. E-mail: josex.gonzalez.gonzalez@intel.com
[‡]Universidad Politécnica de Valencia, Spain. E-mail: jduato@gap.upv.es

Abstract

Cache misses for which data must be obtained from a remote cache (cache-to-cache transfer misses) account for an important fraction of the total miss rate. Unfortunately, cc-NUMA designs put the access to the directory information into the critical path of 3-hop misses, which significantly penalizes them compared to SMP designs. This work studies the use of owner prediction as a means of providing cc-NUMA multiprocessors with a more efficient support for cache-to-cache transfer misses. Our proposal comprises an effective prediction scheme as well as a coherence protocol designed to support the use of prediction. Results indicate that owner prediction can significantly reduce the latency of cache-to-cache transfer misses, which translates into speed-ups on application performance up to 12%. In order to also accelerate most of those 3-hop misses that are either not predicted or mispredicted, the inclusion of a small and fast directory cache in every node is evaluated, leading to improvements up to 16% on the final performance.

1 Introduction and Motivation

The user's view of a shared-memory system is elegantly simple: all processors read and modify data in a single shared store. This makes shared-memory multiprocessors preferable to message-passing multicomputers from the user's point of view. Most shared-memory multiprocessors accelerate memory accesses using per-processor caches. Caches are usually transparent to software through a cache coherence protocol. Directory-based coherence protocols (cc-NUMA multiprocessors) offer a scalable performance path beyond snooping-based ones (SMP designs) by allowing a large number of processors to share a single global address space over physically distributed memory. The main difficulty in such designs is to implement the cache coherence protocol in such an efficient way that minimizes the usually long L2 miss latencies.

Even with non-blocking caches and out-of-order processors, previous studies have shown that the relatively long L2 miss latency found in cc-NUMA multiprocessors constitutes a serious hurdle to performance [21], and, as recently stated by Hill [11], relaxed consistency models do not reduce this long penalty sufficiently to justify their complexity. Thus, there are compelling reasons to examine transparent hardware optimizations.

Several recent research results identify *cache-to-cache transfer misses* (also known as 3-hop misses) to account for more than 60% of the total L2 miss rate in some cases [2][3][7][13][24]. In most cases, cache-to-cache transfer misses occur when the home node has a stale copy of a certain memory line and the most recent copy is dirty in the cache of the processor last wrote it (the owner node). In this situation, as illustrated in Figure 1, the home directory observes the line to be in the *Private* state and forwards the request to the corresponding owner node, which, in turn, sends a copy of the line to the requesting processor as well as a message reporting this to the home directory (which also includes a valid copy of the line for load misses) and properly updates the state of its local copy of the line. Therefore, current cc-NUMA designs place the access to the directory information into the critical path of cache-to-cache transfer misses, which significantly penalizes them compared to SMP designs, and engineering decisions optimizing cache-to-cache transfer misses can be rewarding. As pointed out in [24], these decisions may include faster directory checkup, no speculative read of memory in parallel with directory lookup (which will waste memory bandwidth anyway), faster interconnection network and cache-to-cache transfer support.

In this work we propose and evaluate the use of prediction to convert 3-hop misses into a *new kind* of 2-hop misses. As shown in Figure 1 (right), if the requesting node had been able to "guess" where the single valid copy of the memory line resided, it would have directly sent the miss to the corresponding owner node, removing the access to the directory information from the critical path, as is done in a snooping-based design. As shown in Figure 2, this would bring significant improvements in the final perfor-

0-7695-1524-X/02 $17.00 (c) 2002 IEEE

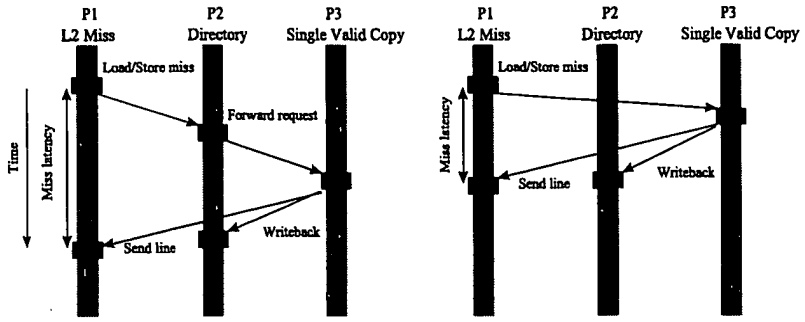

Figure 1: Coherence operations for a 3-hop miss in a conventional cc-NUMA (left) and in a cc-NUMA including prediction (right)

mance (speed-ups[1] up to 26%).

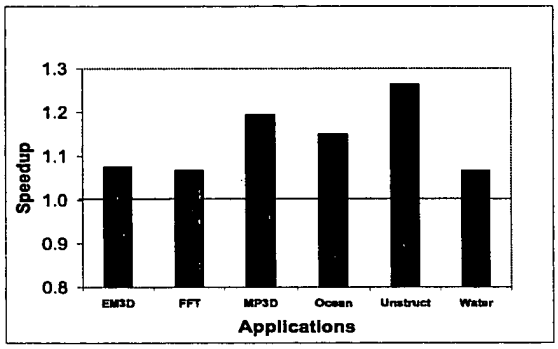

Figure 2: Execution Time Speed-up

Assuming a conventional sequentially consistent cc-NUMA multiprocessor implementing a write invalidate coherence protocol, such as the state-of-the-art SGI Origin 2000 [17], the objective of our proposal is to eliminate the access to the directory information from the critical path of cache-to-cache transfer misses. For this purpose, two main elements are developed: first, a prediction engine able to predict both whether a miss is 3-hop or not and if so, the owner of the line, and second, a coherence protocol designed to support the use of prediction. Our proposal is based on the observation that 3-hop misses usually present a repetitive behavior: they are caused by a small number of instructions and the set of nodes from which the missing instruction receives the corresponding memory line is small (a single node in some cases) and frequently the same. This way, a well-tuned prediction engine could be successfully employed to capture this fact.

Additionally, in order to accelerate the directory accesses for those cache-to-cache transfer misses that are not predicted (or are incorrectly predicted), we analyze the effect of placing a small and fast directory cache into every directory controller to store sharing information only for those lines in the *Private* state.

[1]These speed-up values are obtained with a configuration combining both an almost-oracle predictor and an unlimited directory cache for lines in the *Private* state. See Section 5 for details.

We observe two main contributions of this work:

1. We propose a novel prediction scheme and extend a four-state MESI coherence protocol to support prediction. The use of prediction can significantly reduce the latency of cache-to-cache transfer misses up to a rate of 1.76, which translates into speed-ups on application performance up to 12% and, in general, these results can be obtained including a predictor with a total size of less than 64 KB in every node.

2. We analyze the importance that properly organized directories have to accelerate those 3-hop misses that cannot be predicted (or some of those incorrectly predicted), outlining a directory architecture optimized for cache-to-cache transfer misses. Improvements of up to 16% on the final performance can be obtained combining both prediction and directory caches.

The rest of the paper is organized as follows. Section 3 shows the two-level prediction scheme that we propose. The extended coherence protocol is presented and justified in Section 4. Section 5 presents a detailed performance evaluation of our novel proposals. The related work is given in Section 2. Finally, Section 6 concludes the work.

2 Related Work

Snooping and directory protocols are the two dominant classes of cache coherence protocols for hardware shared-memory multiprocessors. Snooping systems (such as the Sun UE1000 [5]) use a totally ordered network to directly broadcast coherence transactions to all processors and memory. This way, lower latencies than directory protocols are achieved for cache-to-cache transfer misses (for all sharing misses in general). Unfortunately, the energy consumed by snoop requests, snoop bandwidth limitations and the need to act upon all transactions at every processor, make snooping-based designs extremely challenging, especially in light of aggressive processors with multiple outstanding requests. In contrast, directory protocols transmit coherence transactions over an arbitrary point-to-point network to the corresponding home directories which, in turn,

redirect them to the processors caching the line. The consequences are that directory systems (such as the SGI Origin 2000 [17]) can scale to large configurations, but they have higher unloaded latency because of the overheads of directory indirection and message sequencing. Therefore, many research efforts have been focused on studying techniques to reduce the usually long L2 miss latencies that characterize cc-NUMA architectures.

Prediction has a long history in computer architecture and it has proved useful in improving microprocessor performance. Prediction in the context of shared memory was first studied by Mukherjee and Hill, who showed that it is possible to use address-based[2] 2-level predictors at the directories and caches to track and predict coherence messages [19]. Subsequently, Lai and Falfasi modified these predictors to reduce their size and showed how they can be used to accelerate reading of data [16]. Finally, Kaxiras and Young [15] used prediction to reduce access latency in distributed shared-memory systems by attempting to move data from their creation place to their use points as early as possible.

Alternatively, Kaxiras and Goodman [14] proposed and evaluated prediction-based optimizations of migratory sharing patterns (converting some load misses that are predicted to be followed by a store-write fault to coherent writes), wide sharing patterns (to be handled by scalable extensions to the SCI base protocol) and producer-consumer sharing patterns (pre-sending a newly created value to the predicted consumers).

Bilir et al. [4] investigated a hybrid protocol that tries to achieve the performance of snooping protocols and the scalability of directory-based ones. The protocol is based on predicting which nodes must receive each coherence transaction. If the prediction hits, the protocol approximates the snooping behavior (although the directory must be accessed in order to verify the prediction). Performance results in terms of execution time were not reported and the design was based on a network with a completely ordered message delivery which could restrict its scalability. Our work focuses on reducing the latency of 3-hop misses by means of predicting the node that holds the single valid copy of the memory line. We can take advantage of any of the current and future high-performance point-to-point networks and it could be incorporated into cc-NUMA multiprocessors with minimal changes in the coherence protocol.

In [13], Iyer et al. proposed to reduce the latency of the load misses that are solved with a cache-to-cache transfer by placing small directory caches in the crossbar switches of the interconnect to capture and store ownership information as the data flows from the memory module to the requesting processor. However, the fact that special network topologies are needed to keep the information stored in these switch caches coherent represents its main drawback. Our proposal is not constrained to any network topology and it is equally applicable to reduce the latency of cache-to-cache transfer misses caused by load and store instructions.

The Compaq AlphaServer GS320 [7] constitutes an example of cc-NUMA architecture specifically targeted at medium-scale multiprocessing (up to 64 processors). The hierarchical nature of its design and its limited scale make it feasible to use simple interconnects, such as a crossbar switch, to connect the handful of nodes, allowing a more efficient handling of cache-to-cache transfer misses than traditional directory-based multiprocessors by exploiting the extra ordering properties of the switch. On the contrary, our proposal does not require any interconnection network with special ordering.

Finally, caching directory information was originally proposed in [9] and [20] as a means of reducing the memory overhead entailed by directories. In [1], it is proposed a two-level directory architecture as a means of obtaining the performance of a non-scalable full-map directory. Subsequently, we studied the effect that the integration into the processor die of the small first-level directory cache has on the final performance [2]. Additionally, Michael and Nanda [18] proposed to integrate directory caches inside the coherence controllers to minimize directory access time. Our design includes a small and fast directory cache to also accelerate those 3-hop misses that are either not predicted or mispredicted.

3 Predictor Design for Cache-to-Cache Transfer Misses

The first component of our proposal is an effective prediction scheme that allows each node of a cc-NUMA multiprocessor to answer two key questions: first, *is an L2 miss for certain memory line going to be serviced with a cache-to-cache transfer?*, and second, if this is so, *which node is likely to hold the copy of the line?*.

Figure 3 illustrates the anatomy of the prediction scheme we propose and evaluate in this work. The proposed scheme consists of two prediction levels[3]. The *first-level* predictor is mainly in charge of detecting those L2 misses that are probably being satisfied with a cache-to-cache transfer. On the other hand, the purpose of the *second-level* predictor is to provide, for those misses predicted as 3-hop misses, a list of the nodes supposed to have the valid copy of the memory line. Additionally, the *No Predict Table* (*NPT*) is required in order to save the addresses of those memory lines for which a miss caused by a store instruction cannot be predicted. This is done to ensure the correctness of the coherence protocol, as will be discussed in Section 4.

The first-level predictor is an example of an *instruction-based* predictor [14], that is, this level is indexed using the

[2] Address-based stands for predictors whose table is accessed using the effective memory address.

[3] We use the term two-level predictor to mean that our scheme uses two independent prediction tables but, contrary to the traditional 2-level predictors, in our case the information obtained from the first-level table is not used to access the second-level one.

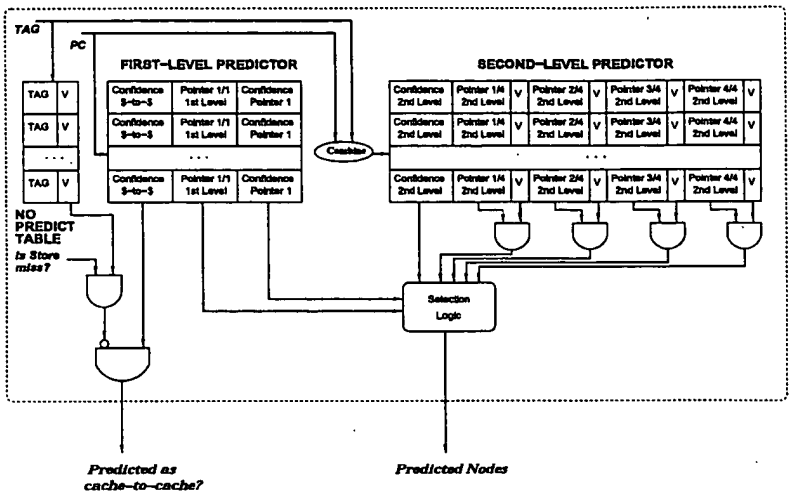

Figure 3: Anatomy of the two-level predictor

PC of the instruction that caused the miss. We have observed that a few static load/store instructions are responsible for the majority of the 3-hop misses. As shown in Figure 3, the *Confidence $-to-$* field, which is implemented as a two-bit saturating counter, along with the information obtained from the *NPT* are used to make the first prediction. Additionally, a pointer to a node is also included in this level (along with its confidence bits). We have observed that, in some cases, the 3-hop misses caused by a certain instruction almost always receive the memory line from the same node. In these situations, the first-level predictor could be used to make both predictions. Each entry in the first-level table needs $(log_2 N + 4)$ bits, for a N-node system.

The first-level predictor is implemented as a non-tagged table and works as follows: initially, all entries in the first-level table have the two saturating counters (*Confidence $-to-$* and *Confidence Pointer 1* fields) initialized to 1. On each L2 cache miss, the predictor is probed and, if the *Confidence $-to-$* counter provides confidence (values of 2 or more) the miss is predicted as 3-hop miss. For store instructions, the address of the line must not be contained in the *NPT*. Later, on the response, predictions are verified, incrementing the *Confidence $-to-$* counter if the miss was serviced with a cache-to-cache transfer or decrementing it otherwise. In case of a 3-hop miss, the *Confidence Pointer 1* counter is also updated, incrementing it when the value stored by the *Pointer 1/1 1st Level* field agrees with the owner of the line or decrementing it otherwise. When this counter reaches 0, the value of the field *Pointer 1/1 1st Level* is changed to the identifier of the sender of the line.

However, in most of cases, a more complex structure is needed to answer the second question. The predictor that provides the list of potential holders of the line (second-level predictor) is accessed using both PC and address information. We have observed that a single static instruction causes 3-hop misses for different memory lines, held by different owners. Therefore, the combination of the PC of the load/store with the effective address provides more accurate information as well as reduces interference. The second-level predictor stores the last four nodes that have had an exclusive copy of the memory line when a miss for this instruction occurred (each one of the *Pointer x/4 2nd Level* fields), with their corresponding valid bits. Again, a two-bit saturating counter (*Confidence 2nd Level* field) is included to reduce mispredictions. Thus, $(4 \times log_2 N + 6)$ bits per entry are needed in this case.

This second-level predictor is also implemented as a non-tagged table and works as follows: initially, all entries in the second-level table have the saturating counter and the valid bits initialized to 1 and 0, respectively. On each L2 cache miss that is predicted as 3-hop miss by the first-level, the second-level predictor is accessed. If the saturating counter (*Confidence 2nd Level* field) gives confidence, the miss is sent to the nodes indicated by those pointers whose valid bits are 1 and to the one indicated by the *Pointer 1/1 1st Level* field whenever this node is not one of the already included (and, of course, if its confidence value is 2 or more). Otherwise, the miss is only sent to the node provided by the first-level predictor (when its counter gives confidence) or it is not predicted and is sent to the home directory as usual. On the responses, predictions are verified and the second-level predictor is updated. The owner is searched in the four pointers and the saturating counter is incremented if it is present, or decremented otherwise. In those cases in which the owner is not contained in the set of pointers, its identifier is also added using one of the pointers that are unused (if any) or replacing the pointer least recently used (when all valid bits are 1).

An important design decision is the maximum number of nodes per prediction. Too few nodes per prediction would cause the second-level predictor to frequently miss for some memory lines (those that are written by several nodes without a defined pattern). However, an excessive

number of nodes wastes network bandwidth and could introduce a significant overhead in the directories as well as in the cache controllers. In our case, we have found that a maximum of five nodes per prediction constitutes a good compromise. One node is obtained from the corresponding first-level predictor entry, while the rest are provided by the second-level predictor.

4 Coherence Protocol Supporting Prediction

Some modifications must be included into the coherence protocol in order to make use of the above prediction scheme. Our starting point is an invalidation-based, four-state MESI coherence protocol as the one included in the SGI Origin 2000 [17]. Two main premises guided our design decisions: first, to keep the resulting coherence protocol as close as possible to the original one, avoiding additional race conditions, and second, to assume sequential consistency [11]. As in [6], we use the following terminology for a given memory line:

- The *directory node* is the node in whose main memory the block is allocated (also known as home node).
- The *exclusive node* is the node that holds the single valid copy of the line.
- The *requesting node* is the node containing the L2 cache that issues a miss for the line.

Requesting Node Operation. When an L2 miss for a certain memory line occurs, the predictor implemented into the cache controller is accessed. If the miss is predicted to be satisfied with a cache-to-cache transfer, a request for the line is sent to the nodes (or node) predicted to have the valid copy of the line (exclusive node). Each request includes the total number of messages sent and a bit identifying it as *predicted*. Otherwise, the request is sent to the directory node, where the miss is satisfied as usual.

Exclusive Node Operation. When a *predicted* request for a certain memory line comes to the cache controller, the line is searched in the L2 cache. If the line is not in the *Exclusive* or the *Modified* states a *nack* message is sent to the directory node notifying that the *predicted* request cannot be satisfied by this node as well as the identity of the requesting node. Otherwise, as it would happen in a non-predicted cache-to-cache transfer miss, the exclusive node immediately sends a copy of the line to the requesting processor as well as an *ack* message indicating this to the directory node (which includes a valid copy of the line for load misses) and properly updates the state of its local copy of the line. Note that for the first case, a *3-hop miss* would be converted into a *4-hop miss*, whereas a new kind of *2-hop miss* is obtained for the second case.

Directory Node Operation. The home directory is responsible for collecting all the responses from the predicted nodes (*ack* or *nack* messages) of a certain prediction. When the first of such responses is received, a buffer entry is allocated[4] and the number of outstanding responses to the prediction is saved. On every additional response, this number is decreased. Once all the responses have been received, one of the following actions will be carried out:

1. In the case of a bad prediction, that is, only *nack* responses have been received, the request is converted into *non-predicted* and is processed as it would be in the normal case.
2. In case an *ack* has been received, two scenarios are possible:
   - 2.1 If the *ack* comes from the expected node, that is, the one codified by the sharing code associated with the memory line, the state and the sharing code must be updated immediately. In those cases in which the memory line has a pending access, the *predicted* request must be processed before that access, since the exclusive node has already serviced the miss. Note that the case in which the line has a pending access when the *ack* is processed is equivalent to having a pending request for a memory line when a *writeback* message for the line is received from the single cache holding the line. This race condition is already considered in the original protocol, so additional changes are not needed to support this case.
   - 2.2 When the *ack* comes from a different node to the one provided by the sharing code associated with the memory line, it means that something preceding this *ack* took place. Therefore, a message is sent to the source of the *ack*, informing that the *ack* could not be observed at that time and a re-send for the message must be performed. The use of retries avoids deadlocks since it ensures that the message from the expected node can find an entry in the directory buffers.

It is important to note that only one *ack* message can be received because, in the case of a prediction hit, there is a single node caching the line and, also, that race conditions for lines in the *Private* state are now solved by the owner cache of the line (not by the directory). Note that mispredictions can only be detected when all the *nack* responses from the predicted nodes have been received. This detection could be done when receiving the first *nack* response if the list of the predicted nodes were included in every message. However, this would increase the total size of each message by three additional bytes to the one already added.

Figure 4 summarizes the previous coherence protocol extended with prediction. As it can be observed, prediction

[4] If a buffer entry is not available a retry message is returned to the sender.

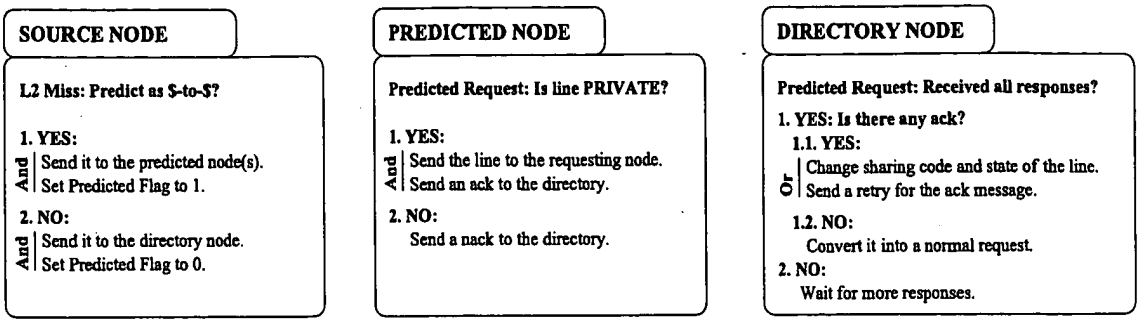

Figure 4: How prediction is included into the original coherence protocol.

could be included in an existing coherence protocol with minimal changes.

There is an additional situation that must be considered in order to preserve the correctness of the coherence protocol. Note that our coherence protocol is based on the fact that at every moment the directory can find the precedence order between all the events related to a memory line in the *Private* state (*predicted* and *non-predicted* requests) as it actually happens. This is possible since the directory always knows where the single valid copy of the line can be found and only a message from such node for the line will be processed. However, when write-write sharing [6] takes place, prediction can make such a precedence order be lost. As an example, assume the next scenario: initially, node *A* is known to have the single copy of a memory line *M*. Then, node *B* wants to write to the line *M*. Obviously an L2 cache miss occurs, and node *B* predicts node *A* as having the single copy of the line. When node *A* sees the *predicted* request, it sends a copy of *M* to node *B*, invalidates its local copy and sends the *ack* message to the home directory. Later on, the processor in node *A* tries to write to the memory line *M* and, again, a miss occurs. Assuming node *A* predicts node *B* as having the line in the *Private* state, the same event sequence would take place. Finally, another node, say node *C* for example, has a write miss for line *M* and predicts node *A* as having the single copy of the line. The problem arises when the home directory sees the second *ack* from node *A* before the very first one. In this situation, the directory assumes the line *M* to have been moved first from *A* to *C*, so that the desired ordering would be lost. This is possible due to: *i)* we assume a point-to-point network without ordering properties, and, *ii)* even with such an ordered point-to-point network, the problem can still occur due to the use of retry messages. Finally, note that if the *ack* from node *B* arrived before the first *ack* from node *A*, a retry message would be returned to *B*.

To avoid the problem, each time that a cache controller receives a *predicted* request that hits in the local L2 cache and that was caused by a store instruction, it looks for a free entry in the No Predict Table described in the previous section. If the *NPT* were full, the request could not be handled as *predicted* and the exclusive node would act as if a miss had taken place. Otherwise, the tag is included into the *NPT* of the exclusive node. If the next miss suffered by this node for the memory line is caused by a store instruction it will not be predicted, forcing the miss to go through the directory to ensure the precedence order. In any case, the miss would free its entry for the memory line in the *NPT*. Observe also that the use of the *NPT* avoids that several predicted accesses prevent indefinitely a non-predict access from obtaining the ownership of the line (that is, livelock situations).

5 Performance Results and Analysis

In this section, we present a detailed performance evaluation of our proposals using extensive execution-driven simulations. First, we present the simulation environment. Next, we analyze the ability of our prediction-based technique to improve performance. Finally, since not all the 3-hop misses can be predicted, we also present results of a directory architecture optimized for 3-hop misses.

5.1 Simulation Environment

We have used a modified version of Rice Simulator for ILP Multiprocessors (RSIM), a detailed execution-driven simulator [12]. RSIM models an out-of-order superscalar processor pipeline, a two-level cache hierarchy, a split-transaction bus on each processor node, and an aggressive memory and multiprocessor interconnection network subsystem, including contention at all resources. The modeled system is a 16-node cc-NUMA that implements a full-map, invalidation-based, four-state MESI directory cache-coherent protocol. Table 1 summarizes the parameters of the simulated system. These parameters have been chosen to be similar to the latencies given in [10] as common values for high-performance multiprocessor systems in the next decade. Second-level caches are assumed to be integrated into the processor chip (as in [10]).

Probing and updating the predictors do not add any cycle. Contrary to the uniprocessor/serial-program context where predictors are updated and probed continuously with every dynamic instruction instance, we only update the prediction history and only probe the predictor to retrieve in-

formation in the case of an L2 miss. Thus, as in [14], we believe that the predictors neither constitute a potential bottleneck nor add cycles to the critical path, because their latency can be hidden from the critical path (for example, by speculatively accessing the predictor in parallel with the L2 cache lookup). Prediction messages are created one-per-cycle.

16-Node System Parameters	
ILP Processor	
Processor Speed	1 GHz
Max. fetch/retire rate	4
Instruction Window	64
Functional Units	2 integer arithmetic 2 floating point 2 address generation
Memory queue size	32 entries
Cache Parameters	
Cache line size	64 bytes
L1 cache (on-chip, WT)	Direct mapped, 32KB
L1 request ports	2
L1 hit time	2 cycles
L2 cache (off-chip, WB)	4-way associative, 512KB
L2 request ports	1
L2 hit time	15 cycles, pipelined
Number of MSHRs	8 per cache
Memory Parameters	
Memory access time	70 cycles (70 ns)
Memory interleaving	4-way
Internal Bus Parameters	
Bus Speed	1 GHz
Bus width	8 bytes
Network Parameters	
Topology	2-dimensional mesh
Flit size	8 bytes
Non-data message size	16 bytes
Router speed	500 MHz
Router's internal bus width	64 bits
Channel width	1 bit
Channel speed	10 GHz
Number of channels	4

Table 1: Base system parameters

With all these parameters, the resulting no-contention round-trip latency of load requests satisfied at various levels of the memory hierarchy is shown in Table 2.

Round Trip Access	Latency (Cycles)
Secondary Cache	19
Local	118
Clean Remote	158 $\backsim$ 218
Cache-to-cache Transfer	224 $\backsim$ 296

Table 2: No-contention round-trip latency of load accesses

Table 3 describes the applications we use in this study. In order to evaluate the benefits of our proposals, we have selected several scientific applications for which cache-to-cache transfer misses constitute an important percentage of the total miss rate (more than 25% in all the cases). *MP3D* and *Water* are from the SPLASH benchmark suite [22], *FFT* and *Ocean* are from SPLASH-2 benchmark suite [23]. *EM3D* is a shared-memory implementation of the Split-C benchmark. *Unstructured* is a computational fluid dynamics application that uses an unstructured mesh. All experimental results reported in this paper are for the parallel phase of these applications. Data placement in our programs is either done explicitly by the programmer or by RSIM which uses a first-touch policy on a cache-line granularity. Thus, initial data-placement is quite effective in terms of reducing traffic in the system.

Program	Size
EM3D	38400 nodes, degree 2, 15% remote, 50 timesteps
FFT	64K Points
MP3D	48000 nodes, 20 timesteps
Ocean	130x130 array, 10^{-9} error tolerance
Unstructured	Mesh.2K, 5 timesteps
Water	343 molecules, 4 timesteps

Table 3: Applications and input sizes

5.2 Predictor Accuracy

The main objective of our prediction-based technique is to directly send those L2 cache misses that are going to be served with a cache-to-cache transfer to the owner of the line. Therefore, two predictions must be carried out: whether or not a certain cache miss is a 3-hop miss and, if so, the identity of the node owning the line. The *two-level* prediction scheme proposed in Section 3 uses the history stored in the first-level table to detect 3-hop misses (*first-level predictor*), whereas the location of the valid copy of the memory line is determined using both the first- and second-level tables (*second-level predictor*). This section analyzes the potential of our two-level predictor assuming an unlimited number of entries in each one of the prediction tables.

Figure 5 illustrates the accuracy of the *first-level predictor*. Over the total number of predictions, it shows the percentage of references that were correctly predicted as 3-hop misses (*Hit*), the percentage of misses that were seen as 3-hop misses but were not (*Miss True*) and the percentage of misses that were incorrectly predicted as non-3-hop misses (*Miss False*). For all the applications but EM3D, our first-level predictor obtains hit rates greater than 80% and, in several cases, this rate is almost 100% (in FFT, Unstructured and Water). EM3D constitutes the only application for which we have observed that the use of address-based prediction (instead of instruction-based) would increase the hit rate.

The accuracy of the *second-level predictor* is presented in Figure 6. In this case, over the number of accesses to this predictor level, it shows the percentage of correct predictions (*Hit Conf*), prediction misses (*Miss Conf*), hits that were not predicted since the confidence counter did not give confidence to the prediction (*Hit No Conf*) and misses that were saved since the counter did not allow the prediction (*Miss No Conf*). As it can be seen, hit rates of more than 60% are obtained for all the applications but MP3D. For this application we have observed that the majority of cache-to-cache transfers occur for a small number

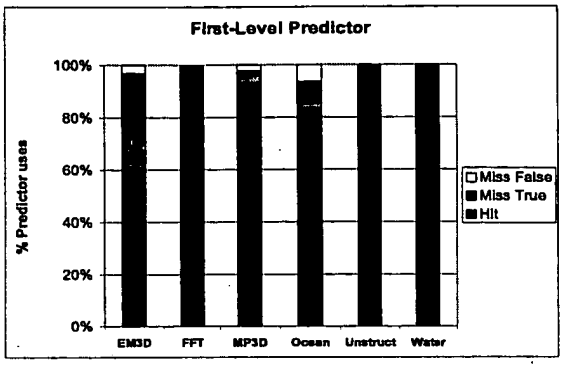

Figure 5: First-level Predictor Accuracy

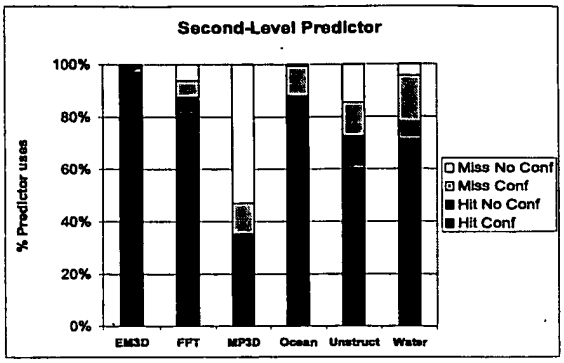

Figure 6: Second-level Predictor Accuracy

of lines that are accessed by all the nodes, which prevents the second level from making predictions (because of the low value of its confidence bits).

Finally, over the total number of 3-hop misses, Figure 7 shows the percentage of those that have been correctly predicted (*Predict*), those that could not be predicted due to the first-level predictor or the second-level one or both not assigning confidence to the prediction (*Non-Confident*), those for which the second-level predictor missed the correct owner (*Miss Predict*) and those that were probed as 3-hop misses but they were not (*Not $-to-$*). The latter is shown starting from 100%, since it corresponds to misses that are not 3-hop misses. As it can be seen, a high percentage of the 3-hop misses can be successfully predicted for Ocean and FFT applications (more than 75%). For EM3D, the hit rate obtained for the first-level predictor negatively influences the percentage of 3-hop misses that can be correctly predicted. The irregular behavior observed in MP3D prevents the second level from predicting a 75% of the 3-hop misses, although the first-level predictor successfully identifies them as being serviced with a cache-to-cache transfer. For this application, the use of confidence bits reduces the number of mispredictions and, then, saves certain misses from wasting bandwidth. Finally, for Water and Unstructured we have found that an important number of 3-hop misses (13% and 19%, respectively) are not predicted due to the high number of store misses for which an entry in the *NPT* was found. The reason is the significant amount of false sharing observed in these applications. Note also that for all the applications the number of 3-hop misses that could not be predicted (*Non-Confident*) exceeds those that were incorrectly predicted (*Miss Predict*). *Non-Confident* and *Miss Predict* cases will be considered again in Section 5.4. Finally, the percentage of misses incorrectly predicted as 3-hop (*Not $-to-$*) is very low, which demonstrates the high accuracy of the first-level predictor.

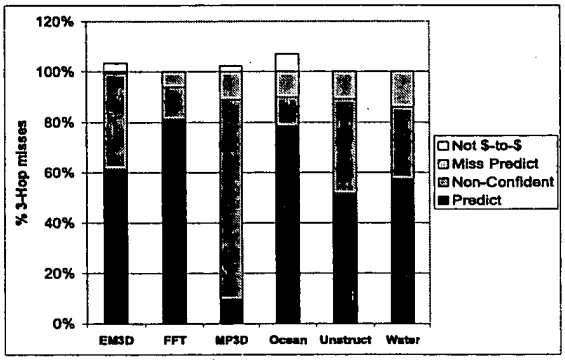

Figure 7: Percentage of 3-hop misses predicted

5.3 Performance Analysis

In this section we analyze quantitatively the performance benefits of our proposal. First, we study how prediction affects the average latency of 3-hop misses, then the effect on the average latency of load and store misses is presented, and finally, execution time speed-ups are also reported. For all cases, we compare a *base* configuration, which does not use prediction, a configuration with an almost-oracle predictor (*AOP*), which gives us an approximation of the maximum benefit that could be obtained, and two configurations using the two-level predictor: *UL-2Level*, for which each prediction table has an unlimited number of entries (and for which accuracy results were presented in the previous section) and *L-2Level*, that limits the size of the prediction tables.

Reductions in the latency of 3-hop misses as well as load and store misses are seen in terms of the *reduction rate*, which is calculated as:

$$\text{Reduction rate} = \frac{\text{Base Average Latency}}{\{AOP,\ UL\text{-}2Level,\ L\text{-}2Level\}\ \text{Average Latency}}$$

The *AOP* predictor used in this work accesses the directory information on every L2 cache miss, to determine if the line is in the *Private* state and, if so, which node is caching the line, and directly sends the miss to the corresponding node. We have modified RSIM to allow nodes suffering an L2 cache miss to directly access the corresponding directory entry without spending any cycle. However, this only constitutes an approximation of the oracle predictor behavior since mispredictions are still possible. For example, when two different nodes make a predic-

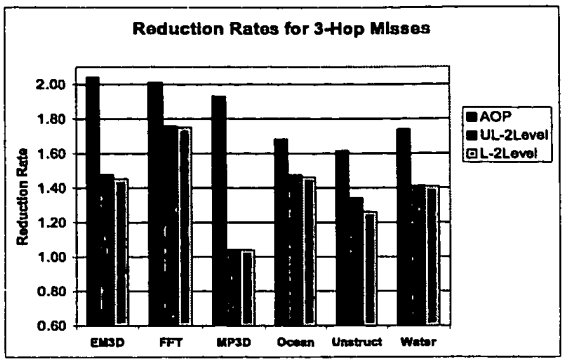

Figure 8: Reduction Rates for 3-Hop Misses

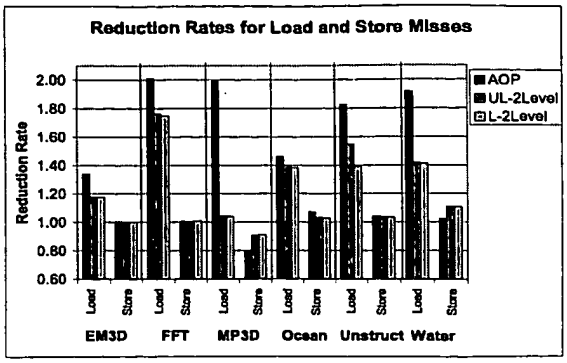

Figure 9: Reduction Rates for Load and Store Misses

tion at the same time for the same memory line, one of them will miss. However, these situations occur infrequently so that more than 90% of the predictions were correct for all the applications. Moreover, since the *AOP* predictor sends a single message per prediction, misprediction penalty is always kept very small.

The *L-2Level* predictor constitutes an example of how a "realistic" implementation of the *UL-2Level* predictor would behave. The total size of this predictor is kept below 64 KB, for which there are 2K entries for the first-level prediction table (total size of 2 KB), 16K entries for the second-level prediction table (total size of 48 KB) and 128 entries for the *NPT* (total size of 512 Bytes), which is enough, since we have observed that a small number of entries are used in this table. The first-level table is indexed directly using ten least significant bits of the PC of the instruction missing in the L2 cache. The access to the second-level table is carried out from the result of computing the XOR between bits from 5 to 18 of the missing address and bits from 2 to 15 of the PC. As in [8], we use XOR-based placement to optimize the use of the entries in the second-level table. Note that both prediction tables are non-tagged and aliasing can occur. Finally, due to its small number of entries, the *NPT* is organized as a totally associative buffer structure.

Application	AOP	UL-2Level	L-2Level
EM3D	1.00	1.07	1.47
FFT	1.00	1.02	1.32
MP3D	1.00	3.58	3.61
Ocean	1.00	1.06	1.11
Unstructured	1.00	3.11	3.49
Water	1.00	3.59	3.61

Table 4: Number of nodes included per prediction

Figure 8 presents how the use of prediction accelerates 3-hop misses in *AOP*, *UL-2Level* and *L-2Level* configurations with respect to the base system, whereas Table 4 shows the average number of nodes included in each prediction. As can be observed from *AOP* results, prediction has the potential to significantly improve 3-hop misses for all the applications (3-hop miss average latency is reduced by half for EM3D and FFT and by a rate of more than 1.6 in all cases). In practice, the latency reduction that could be reached with a non-oracle predictor is lower since not all the 3-hop misses can be correctly predicted and, for some predictions, messages to several nodes must be sent. However, these benefits are still very important for all applications but MP3D when using the *UL-2Level* predictor (reduction rates ranging from 1.34 for Unstructured to 1.76 for FFT) and, what is more important, they could be obtained with a "realistic" configuration (for all the applications but one *UL-2Level* and *L-2Level* predictors obtain the same results). The exception is Unstructured for which a slightly lower reduction rate is found for *L-2Level* (from 1.34 to 1.26). Remember from last section that for MP3D the two-level predictor was unable to predict the majority of 3-hop misses, so performance benefits cannot be expected for this application. Finally, the non-tagged nature of the *L-2Level* predictor makes some of its entries be shared between different predictions which, as shown in Table 4, slightly increases the average number of nodes per prediction when compared to the *UL-2Level* scheme.

Application	Load Misses	Store Misses	Total Misses
EM3D	36.49%	0.00%	26.73%
FFT	99.41%	0.00%	54.35%
MP3D	95.34%	4.95%	50.08%
Ocean	77.84%	5.28%	43.98%
Unstructured	80.38%	52.63%	62.92%
Water	91.50%	41.54%	61.03%

Table 5: Percentage of 3-hop misses found in load, store misses and in the total misses

The important benefits found for 3-hop misses also lead to reductions in the average latency of load and store instructions. As can be observed from Table 5, for all the applications but EM3D the most important fraction of the load misses is serviced with a cache-to-cache transfer which, as shown in Figure 9, translates into significant reductions on load miss latencies when compared to the base system. Again, *UL-2Level* and *L-2Level* obtain virtually identical improvements for all the applications but Unstructured (reduction rates of 1.18 for EM3D, 1.76 for

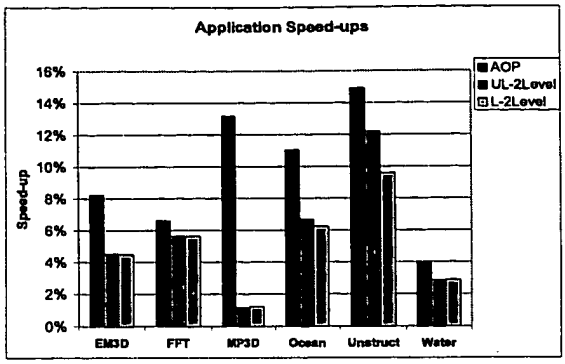

Figure 10: Application speed-ups

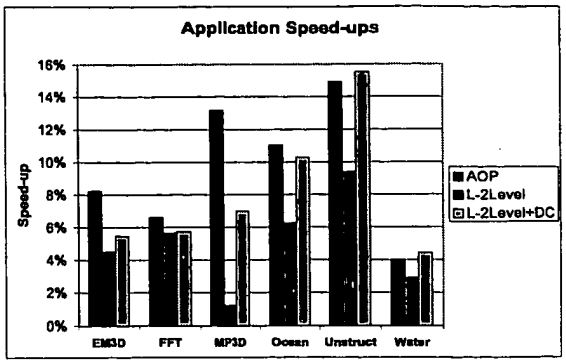

Figure 11: Application speed-ups obtained when directory caches for 3-hop misses are used

FFT, 1.40 for Ocean and 1.42 for Water). For this application reduction rates of 1.55 and 1.36 are obtained for *UL-2Level* and *L-2Level*, respectively. On the other hand, the benefits found for store misses are not so significant (less than 1.10 for all cases) and even a small slow-down is observed for MP3D. These results can be expected for EM3D, FFT, MP3D and Ocean due to, as illustrated in Table 5, a very small percentage of the 3-hop misses was caused by store misses (0% in some cases). On the contrary, the fraction of store misses serviced with a cache-to-cache transfer is substantial for Unstructured and Water. However, and as previously seen, the false sharing experienced in these applications forces the majority of the store misses not to be predicted, explaining the low potential obtained in Figure 9 for these applications. Therefore, two-level predictors could be simplified (eliminating the need of having the *NPT*) without significantly hurting the final performance by not predicting store misses. Note also that for all the applications but MP3D, *UL-2Level* and *L-2Level* configurations obtain improvements near to those found for *AOP*.

The ultimate metric for application performance is the execution time. Figure 10 shows the speed-ups in execution time for *AOP*, *UL-2Level* and *L-2Level* configurations normalized with respect to the base system. We find that, as expected, negligible improvements are obtained for MP3D when both *UL-2Level* and *L-2Level* predictors are used (speed-up of 1%), although important benefits could be obtained (speed-up of 13% for *AOP*). For the rest of the applications, *UL-2Level* and *L-2Level* configurations reach more than 50% of the performance benefits obtained for *AOP*. For EM3D, FFT, Ocean and Water speed-ups of 5%, 6%, 7% and 3%, respectively, are found for *UL-2Level* and *L-2Level*, while for Unstructured the improvements reached differ in a 2% (speed-ups of 12% for *UL-2Level* and 10% for *L-2Level*).

5.4 Including a Directory Cache into the Final Design

One way to also accelerate non-predicted and some of the mispredicted 3-hop misses is by reducing the time needed to obtain the identity of their destination node. For this, we study the case of adding a small and fast directory cache to every directory controller. This directory cache stores sharing information only for those lines in the *Private* state. Thus, each one of its entries contains only a 1-pointer sharing code to keep the identity of the single node caching the line (as well as some tag information). The latency of the directory cache is assumed to be 10 cycles (1 directory cycle), while 70 cycles must be spent when accessing to the main directory (which is the latency of the main memory).

In this way, those 3-hop misses that could not be predicted and for which the directory cache contains their corresponding directory entries will be quickly routed to their owner node, saving the cycles needed to access the slow DRAM directory. Observe that, on the contrary, those mispredictions for which several messages are sent cannot take any advantage from the use of the directory cache, since the miss is detected once all the *nack* responses from the predicted nodes arrive to the directory. For predictions with more than one message involved, we assume that the access to the main directory begins when the *ack* or the first *nack* for the *predicted* request reaches the directory, and finishes when receiving the last one.

Figure 11 shows how the improvements obtained with the *L-2Level* predictor could be even increased if a small directory cache were used in every node (*L-2Level+DC*). Results for *L-2Level* and for *AOP* are also included for comparison purposes. The directory caches modeled in these simulations are fully associative, 512-entry structures which use a LRU replacement policy[5]. As derived from Figure 11, adding a directory cache to the final design would bring performance benefits close to those obtained with the *AOP* predictor for all the applications but MP3D. Even these benefits outperform the ones observed for the *AOP* case in Unstructured and Water. Remember that an important percentage of the 3-hop misses observed in these applications was caused by a store instruction for which an entry in the *NPT* table was found and, consequently, that

[5]Practical implementations can be set-associative, achieving similar performance at lower cost [18].

could not be predicted even with the *AOP* scheme. On the other hand, for MP3D prediction was shown to bring negligible improvements in execution time, so that the speed-up of 7% is mainly due to the use of the directory cache.

6 Conclusions

Several recent studies have observed cache-to-cache transfer misses to constitute an important fraction of the total miss rate (more than 60% in some cases), so that optimizations to reduce the usually long latencies associated with these misses have become the subject of important research efforts. In this work, we propose the use of prediction to directly send 3-hop misses to the corresponding node where the single valid copy of the line resides. This would eliminate the significant number of cycles needed to access the directory information from the critical path of 3-hop misses.

The prediction-based technique proposed in this work consists of two main components. The first one is a novel two-level prediction scheme achieving high hit rates and the second is a coherence protocol, similar to the one used in the SGI Origin 2000, properly extended (with minimal changes) to support the use of prediction. The use of prediction can significantly reduce the latency of cache-to-cache transfer misses up to a rate of 1.76, which translates into speed-ups on application performance up to 12% and, in general, these results can be obtained including a predictor with a total size of less than 64 KB in every node.

In addition, we found that a substantial number of 3-hop misses remained non-predicted (or mispredicted) and showed how including in every node a first-level directory cache made up of a small number of 1-pointer entries helped to increase the benefits of using prediction (speed-ups on application performance up to 16%).

Additional optimizations for 3-hop misses could be derived from the results of this work. For example, the high hit rates observed for the first-level predictor suggest that small predictors could be introduced in every node to avoid, when detecting a 3-hop miss, the speculative read of memory that, otherwise, would be done in parallel with the access to the directory (which wastes memory bandwidth).

Acknowledgments

This research has been carried out using the resources of the Centre de Computació i Comunicacions de Catalunya (CESCA-CEPBA) as well as the SGI Origin 2000 of the Universitat de Valencia. This work has been supported in part by the Spanish CICYT under grant TIC2000-1151-C07-03. José Duato is supported in part by a fellowship from the Fundación Séneca (Comunidad Autónoma de Murcia, Spain).

References

[1] M. E. Acacio, J. González, J. M. García and J. Duato. "A New Scalable Directory Architecture for Large-Scale Multiprocessors". *Proc. of the 7th Int'l Symposium on High Performance Computer Architecture (HPCA-7)*, pp. 97–106, January 2001.

[2] M. E. Acacio, J. González, J. M. García and J. Duato. "A Novel Approach to Reduce L2 Miss Latency in Shared-Memory Multiprocessors". *Proc. of the 16th Int'l Parallel and Distributed Processing Symposium (IPDPS'02)*, April 2002.

[3] L. A. Barroso, K. Gharachorloo and E. Bugnion. "Memory System Characterization of Commercial Workloads". *In Proc. of the 25th Int'l Symposium on Computer Architecture (ISCA'98)*, pp. 3–14, June 1998.

[4] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill and D. A. Wood. "Multicast Snooping: A New Coherence Method Using a Multicast Address Network". *Proc. of the 26th Int'l Symposium on Computer Architecture (ISCA'99)*, pp. 294–304, May 1999.

[5] A. Charlesworth. "Extending the SMP Envelope". *IEEE Micro*, 18(1):39–49, Jan/Feb 1998.

[6] D. E. Culler, J. P. Singh and A. Gupta. *"Parallel Computer Architecture: A Hardware/Software Approach"*. Morgan Kaufmann Publishers, Inc., 1999.

[7] K. Gharachorloo, M. Sharma, S. Steely and S. V. Doren. "Architecture and Design of AlphaServer GS320". *Proc. of International Conference on Architectural Support for Programming Language and Operating Systems (ASPLOS IX)*, pp. 13–24, November 2000.

[8] A. González, M. Valero, N. Topham and J. M. Parcerisa. "Eliminating Cache Conflict Misses through XOR-Based Placement Functions". *Proc. of the Int'l Conference on Supercomputing (ICS'97)*, pp. 76–83, 1997.

[9] A. Gupta, W.-D. Weber and T. Mowry. "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes". *Proc. Int'l Conference on Parallel Processing (ICPP'90)*, pp. 312–321, August 1990.

[10] L. Gwennap. "Alpha 21364 to Ease Memory Bottleneck". *Microprocessor Report*, pp. 12–15, October 1998.

[11] M. D. Hill. "Multiprocessors Should Support Simple Memory-Consistency Models". *IEEE Computer*, 31(8):28–34, August 1998.

[12] C. J. Hughes, V. S. Pai, P. Ranganathan and S. V. Adve. "RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors". *IEEE Computer*, 35(2):40–49, February 2002.

[13] R. Iyer, L. N. Bhuyan and A. Nanda. "Using Switch Directories to Speed Up Cache-to-Cache Transfers in CC-NUMA Multiprocessors". *Proc. of the 14th Int'l Parallel and Distributed Processing Symposium (IPDPS'00)*, pp. 721–728, May 2000.

[14] S. Kaxiras and J. R. Goodman. "Improving CC-NUMA Performance Using Instruction-Based Prediction". *Proc. of the 5th Int'l High Performance Computer Architecture (HPCA-5)*, pp. 161–170, January 1999.

[15] S. Kaxiras and C. Young. "Coherence Communication Prediction in Shared-Memory Multiprocessors". *Proc. of the 6th Int'l High Performance Computer Architecture (HPCA-6)*, pp. 156–167, January 2000.

[16] A. C. Lai and B. Falsafi. "Memory Sharing Predictor: The Key to a Speculative DSM". *Proc. of the 26th Int'l Symposium on Computer Architecture (ISCA'99)*, pp. 162–171, May 1999.

[17] J. Laudon and D. Lenoski. "The SGI Origin: A ccNUMA Highly Scalable Server". *Proc. of the 24th Int'l Symposium on Computer Architecture (ISCA'97)*, pp. 241–251, June 1997.

[18] M. M. Michael and A. K. Nanda. "Design and Performance of Directory Caches for Scalable Shared Memory Multiprocessors". *Proc. of the 5th Int'l Symposium on High Performance Computer Architecture (HPCA-5)*, pp. 142–151, January 1999.

[19] S. S. Mukherjee and M. D. Hill. "Using Prediction to Accelerate Coherence Protocols". *Proc. of the 25th Int'l Symposium on Computer Architecture (ISCA'98)*, pp. 179–190, July 1998.

[20] B. O'Krafka and A. Newton. "An Empirical Evaluation of Two Memory-Efficient Directory Methods". *Proc. of the 17th Int'l Symposium on Computer Architecture (ISCA'90)*, pp. 138–147, May 1990.

[21] V. S. Pai, P. Ranganathan, H. Abdel-Shafi and S. Adve. "The Impact of Exploiting Instruction-Level Parallelism on Shared-Memory Multiprocessors". *IEEE Transactions on Computers*, 48(2):218–226, February 1999.

[22] J. Singh, W.-D. Weber and A. Gupta. "SPLASH: Stanford Parallel Applications for Shared-Memory". *Computer Architecture News*, 20:5–44, March 1992.

[23] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". *Proc. of the 22nd Int'l Symposium on Computer Architecture (ISCA'95)*, pp. 24–36, June 1995.

[24] Z. Zhang. "Architectural Sensitive Application Characterization: The Approach of High-Performance Index-Set (HP-Set)". *Technical Report HPL-2001-75, HP Laboratories Palo Alto*, March 2001.